

Interactive Systems on the Road: Development of Vehicle User Interfaces for Failure Assistance

Geert Houben

Jan Van den Bergh

Kris Luyten

Karin Coninx

Limburgs Universitair Centrum
Expertise Centre for Digital Media
Universitaire Campus
B-3590 Diepenbeek, Belgium

{geert.houben,jan.vandenbergh,kris.luyten,karin.coninx}@luc.ac.be

Abstract

In-vehicle interactive systems are usually specialized systems that rely on a particular set of hardware and are designed to support the driver in reaching a destination with his/her vehicle. Since the importance of these systems, especially when they are built to support the driver while driving, a lot of research already has been done concerning usability and safety. In contrast, interactive systems that support the automotive after sales market have generated much less attention. Here, other aspects need to be emphasized such as the failure type and complexity, means for information visualization and communication, and connectivity. We present an overview of the different interaction devices used for a specific scenario of the European Project MYCAREVENT. Integration between personal mobile devices and the environment of the car is envisioned, which will eventually lead to a multi-channel system where the most appropriate device with the best connection can be selected to interact with the user and the car. We will focus on different methodologies to create consistent and reliable user interfaces for multiple in-car interaction devices. A custom notation is introduced that supports the creation of context-sensitive interactive systems suitable to support the user in the given situation: a car breakdown. One part of the context that can be taken into account is the connectivity of the personal mobile device, since this influences the amount of information that is accessible for the user.

1 Introduction

During the last few years, the automotive sector offers a lot of services to its end-user, ranging from safety-improving systems over entertainment to navigation applications.

The communication possibilities of most mobile systems allow to integrate these systems with the car to support the driver in different tasks. A typical example that

is used nowadays is a Personal Digital Assistant (PDA) equipped with navigation software that can be easily integrated in the car environment [3, 7]. A lot of research in this area has been done, especially concerning usability and safety purposes. After all, most of these services are used *while driving* the car and it requires an approach where the interface does not have the full attention of the driver [4, 10].

Mobile devices, like PDAs and cellphones, have become common in use. Their relative low cost, combined with their computing power and communication abilities, make them suitable interaction devices to be used in our everyday tasks. We can take advantage of these devices in case of a car failure to assist the driver while solving the problem. In the European Project MYCAREVENT (MCE) scenario's are being developed that specify how a driver can repair a problem her- or himself assisted by different interconnected systems. Personal mobile devices can play an important role in these scenario's. Previously, when trying to repair these car failures, there were no interactive services or systems available in the car to support the repair process. The only solution was to call a road-side technician or look for a nearby garage where the car can be repaired. In this article we will focus on the necessary basis to use mobile devices to support the driver in case of a car breakdown.

When providing a failure assistance system to the driver, several important issues have to be taken into account during development: handling complexity, clear visualization of information, being failsafe and reliable, ensuring connectivity and security, . . . Much less attention is paid to these systems and to the computerization of the automotive after sales market. We focus on different methodologies for creating user interfaces for these systems. The introduction of innovative interactive applications creates new ways of collaboration among car manufacturers, workshops, road assistance services and the end-user.

This paper is structured as follows: Section 2 gives an overview of the devices that will be used to guide the user in case of a car breakdown. Next, Section 3 discusses several design methodologies suitable for user interface engineering in the given context. Section 4 describes a process in which detection and reparation of the driver's car is performed from a user interface engineering point of view. Section 5 introduces a custom notation to design an interactive system that can accommodate to the requirements imposed by the MCE scenario discussed in Section 4. Finally, Section 6 draws up the conclusions of this paper.

2 In-Car Interaction Devices

Nowadays we notice that the resources of mobile systems as well as their usage is remarkably increasing. The mobile world offers a huge amount of devices with noticeable diversity in screen size, computing power, amount of memory, interaction handling, connectivity, The purpose of mobile devices is moving from a simple apparatus with a single user goal to a multi-purpose multimedia device, ready to use it for all kinds of mobile interactions.

Also in the MCE project we will use mobile devices as the tool of the user to interact with the car and the MCE system. In Section 4 we work out a MCE scenario; the service under consideration in this section is called "driver self-help". In this scenario the contribution of the driver is an essential part during the cycle of detection and reparation a specified car-related problem. This contribution not only covers the determination of the fault, but also the possibility to fix the problem by the driver himself. We will use this example as the leitmotiv in the text and we will map a custom notation on this example, as explained in Section 3.

Because these days mobile devices are almost completely integrated in the personal environment and life of the user, we can rely on it as the center of the service. While developing such services, some issues have to be addressed: reliable applications have to be provided (software crashes are barely allowed), connections have to be reliable too, and authentication and security of the user information is necessary. Cars have some critical electronic and mechanical parts that need be treated with care. Therefore, failsafe applications equipped with up-to-date content information to support the driver have to be guaranteed.

To steer the actions of the user, "in-car" interaction devices are used to lead him through the several necessary steps. These devices are the link between the detected fault in the car and the Service Portal, the MCE system "solving" the problem. These devices have typical properties, interfaces and interaction methods, because they

are used in other situations and ways the user is used to. We assume some interactive actions between the driver and her/his device: providing diagnostics based on perceived and described symptoms to the MCE system, receiving solutions from the system, communicating with the On Board Diagnosis (OBD) device etc.

The mobile devices we will consider in our classification are Personal Digital Assistants (PDA), smartphones and mobile phones. We focus on their functionality and usability. Only *wireless* connectivity between the mobile device and the OBD and between the mobile device and the Service Portal is assumed. Some criteria, by which we will test the devices, are fixed: (1) graphical capabilities, (2) interaction methods, (3) communication possibilities, (4) the support of the type of content that has to be provided, and (5) the usability of application development. Two sorts of communication have to be provided in the scenario: long range communication (LRC) with the Service Portal and short range communication (SRC) with the car.

Mobile phones or cell phones are the smallest devices we consider. They have limited memory capacity and processor speed, and the screen is pretty small. (1) Cell phones does not have advanced graphical possibilities, so the interface is mostly textual and *form-based*. (2) The input is provided by number buttons and soft buttons. (3) Most modern cell phones support various way of data communication. Mostly used for human-human communication is the digital GSM technology (Global System for Mobile communications). To provide failure information to a call center operator, this call functionality is necessary. Also infrared, Bluetooth (SRC), GPRS and UMTS (LRC) are among the most popular supported protocols and technologies. GPRS is especially useful for transferring data, it is a packet switched technology (in contrast with GSM, which is circuit switched). (4) Due to the phone limitations, only call functions and textual or simple graphical information can be provided. Most of the users have a cell phone, so user interactions became established over the last few years. (5) Besides the limited system properties, it is difficult to implement applications for mobile phones. The software is printed on the chip and it is not easy to change or modify it. A possible way to develop programs for them is with Java (J2ME).

Personal Digital Assistants (PDAs) are small, portable computing devices which are particularly used for carrying information like an addressbook, agenda, phone number list, emails etc. (1) They have high

resolution screens and much more graphical capabilities. PDAs can be used for a lot of multimedia purposes, like playing movies and music, taking pictures, receiving emails etc. (2) The interaction is handled with a touchscreen and stylus and some simple (hardware) buttons. (3) There are some PDAs with phone functions, but they are not widespread. Bluetooth, Wireless LAN and infrared (SRC) are the most common ways for wireless communication. (4) PDAs are very suitable for visualizing graphical information. Short range communication is provided, long range with PDAs is not obvious. (5) The most common operating systems for PDAs are Microsoft Pocket PC, Microsoft Windows Mobile, Palm OS and Linux. There are good Integrated Development Environments (IDEs) for these operating systems, so writing software for them is straightforward and strongly related to programming for desktop environments (e.g. .NET Compact Framework for MS Pocket PC).

Smartphones combine the possibilities of PDAs with those of cell phones. (1) The graphical possibilities are high too, related to a PDA, but their screen size is smaller. Also the screen resolution is not as high as PDA screens. (2) Interaction is handled by number buttons and softbuttons, added with a little joystick or even a touchscreen and stylus. (3) Smartphones have more wireless communication options than PDAs, like GSM and GPRS (LCR), but also Bluetooth and infrared (SRC). (4) They are also multimedia-related: you can take pictures, send images, play games, receive emails, keep an agenda etc. Graphical visualization of data is possible, and they provide long range as well as short range communication technologies. (5) Widely used on smartphones is the Symbian operating system, which is suitable to write applications for (in Java as well as in C++). Few smartphones run on Windows Mobile OS. Application development on smartphones is not straightforward if one is used to desktop PC programming.

It is obvious that information for repairing cars has to be visualized as clear as possible, so it will be better to have a large screen with a high resolution than a small one with a low resolution. In this case, visualization of data is more important than a fast connection with the OBD or the Service Portal. On a small, textual display it is difficult to show a lot of important data without losing the focus on the arrangement of information. Because repair information is in fact just convenient if it is combined with visual info, for example pictures or movies,

graphical capabilities are no exaggerated luxury. A PDA or smartphone seems to be the best solution to support this needs.

Because smartphones are focused on combining the advantages of cell phones and PDAs, they have the graphical capabilities and communication possibilities to fulfill our conditions. Most of the smartphones support GSM, GPRS and Bluetooth technology, and have a high-resolution color screen. However, PDAs have a better support for programming applications and user interfaces and are more suitable for playing movies, showing pictures or running memory intensive applications. They provide easier and faster user interactions (touchscreen and stylus) related to most of the smartphones (hardware buttons). However, GSM technology is not yet available in common PDA devices.

3 Design Methodologies

This section provides an overview of different design methodologies that can support the interface engineer to develop user interfaces for failure assistance. The design methodologies that are used here should support different aspects that are typical for the situation. E.g. the driver, whose car is suffering some type of defect, is probably in a very stressful situation. Furthermore, if a driver can fix the failure without help of a human expert, the information that is provided to guide the driver must be correct and complete, otherwise this could lead to a new failure or worse. Therefore, the design of the interactive part of the system should give special attention to the following aspects:

- Correct behavior at all times
- Robustness in all possible situations
- Context-sensitive

Correct behavior and robustness are long standing pillars for these type of applications. Context-sensitivity is a new aspect: diverse and easily accessible sensors provide the application with information from its surroundings (e.g. location of the car, heat of the car engine, available communication networks, outside temperature,...). Particularly in the cases defined by the MCE partners, such as “driver self-help”, context-sensitive interactive systems can play an important role. The design methodologies used to create user interfaces for the devices are introduced in Section 2.

Formal user interface specification techniques are well suited to take into account these aspects. Formal methods are already widely used in traditional software engineering approaches [11] where safety, correctness and reliability are important. A formal description of software

ensures the reaching of predefined requirements, because one can prove the software will work as specified. For the same reason, formal methods can aid the user interface designer to create a robust interface that shows correct behavior at all times. [1] shows a user interface component can be the subject of a formal verification process. [2] present a design framework based on the Interactive Cooperative Object (ICO) formalism to create safety critical interactive systems. [1] and [2] are two examples of formal approaches for designing an interactive system where safety (and thus correct behavior) is an important issue.

Besides formal specifications, model-based user interface development (MBUID) [8] is a design methodology that is applicable in the aforementioned situations where multiple device, multiple users and multiple contexts have to be taken into account. At the basis of MBUID a set of models is used. Formal methods are often used as part of the MBUID approach to describe the different models, although this is not a requirement. A model can be informally defined as a non-empty set with elements and a set of relations specified between these elements. It gathers and relates information about the specific concept the final interface should reflect. A model provides an abstraction of this concept: it hides the low-level details while preserving the important ones. It typically focuses on the important characteristics that make up the interface concept; the specification of low-level details is postponed to a later stage in the design process. The Task Model is probably the most well-known model: it describes the activities, tasks and sub-tasks that are performed to reach a goal from a user's perspective. For example [9] shows task models are suitable to be used for safety-critical systems. In other work we showed context can be integrated in task models [5]: one possible use of this is to allow the user interface designer to make a difference between the availability of a communication network or no available communication network.

Besides the discussed methods that allow the user interface designer to emphasize robustness, correctness and context-of-use, another methodology that is well-established is the traditional software engineering. Traditional software engineering uses the Unified Modeling Language¹ as its standard notation: a language that is widely used in different software processes (e.g. the Rational Unified Process) but lacks support for user interface engineering. This is fortified by the output of the IFIP working group 2.7 / 13.4 on User Interface Engineering², whose goal is to bridge the gap between Human Computer Interaction and Software Engineering. Sec-

tion 4 shows a way to design user interfaces for wireless-connected systems while providing a smooth integration with UML.

4 Driver Self-help

In this section we introduce a specific scenario on which we will apply a notation we developed to support the design of the user interface. First of all we will describe the scenario, based on the "driver self-help" scenario of the MCE project. An overview can be found in Figure 1.

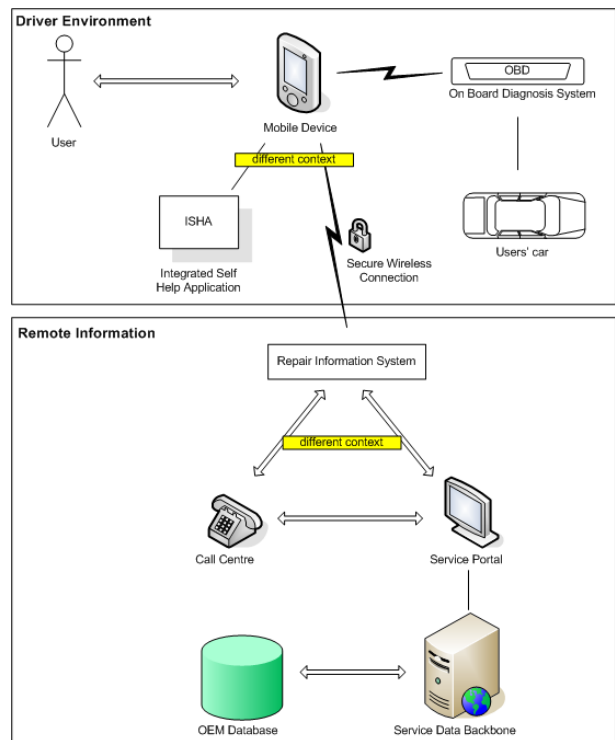


Figure 1: Overview of the "driver self-help" scenario

The core idea of the MCE project is the failure assistance of a user's vehicle. By setting up a car manufacturer database and a Service Portal to communicate with the user, failure related information, based on the given input, is supplied to the driver. Also road-side technicians or workshops can use this remote "repair information" system. In this text we will only focus on the role of the driver as an actor in the process. The scenario described in this section is based on a MCE use case and it serves as an example in this paper. Our goal is not to provide a complete scenario.

Several ways for detection and/or reparation failures in cars are defined: detection and reparation by vehicle software, by the driver himself, by a road-side technician or by a workshop technician. Detecting the car problem should be left completely to car electronics: sensors in

¹<http://www.uml.org/>

²<http://www.se-hci.org>

the vehicle determine the fault(s), after which they are bundled in the On Board Diagnosis (OBD) system.

However, when a breakdown situation occurs, these self-diagnosis abilities are limited to the data collected by the sensors. This brings some problems along. First of all, it is virtually impossible to cover all the conceivable in-car faults. Secondly, when the sensors themselves are subject to defects no relevant data can be obtained from that part of the vehicle. This results in insufficient input for failure reparation. For this reasons, besides being the principal user of the system, the driver is also required as a source of information.

The car driver has the possibility to give some input concerning possible car failures. In the scenario defined and used in this paper the user will do this with her/his mobile device. The input from the OBD device is combined with input provided by the user. After detecting the failure(s), they are “translated” in a uniform language on the mobile device. On the device, an Integrated Self Help Application (ISHA) is running and will be firstly consulted by providing the failure data. This application is an autonomous solution that is independent of the Service Portal and works in the driver’s personal environment. As soon as the problem is analyzed and a solution can be given to the driver, he/she will be guided through the reparation process by the system by means of the available interaction devices (see Section 2).

If the ISHA is not able to solve the problem, other services have to be addressed. At this point, the MCE Service Portal is used. The problem data is analyzed and it is checked for possible solutions which can be supplied for the specified fault information. If there are solutions available, a solution picking list is created and sent back to the driver’s mobile device. If the problem is not fully covered or solved, we have to go one step further. The last node in the solution providing chain is the Call Center Operator (CCO).

The CCO is also part of the remote repair information system. When the computer system of the Service Portal is not able to solve the car problems, we have to omit the computer as a reparation manual. The use of human intelligence is the only appropriate approach left attempting to solve the vehicle defects. The driver is connected to the call center and he can provide extra failure information to the operator in real-time. This operator can use his own knowledge, if necessary completed with information from the Service Data Backbone to give the user the right repair instructions. If this approach is also not sufficient, an expert has to come on the spot to examine and/or repair the car.

5 Design and implementation support

In this section, the described scenario of Section 4 is used as an example to illustrate a user interface modeling notation that is closely related with traditional software modeling notations. We propose to develop one application for the mobile device of the driver that is represented by different interfaces, depending on her/his position in the detecting and repairing process. The interface is changing due to these context changes. Also the user profile can be taken into account when generating the user interface.

Figure 2 shows an activity model for the scenario using the notation discussed in [12]. Notice that different types of actions (user, interaction, system and environment) are defined in the activity model and the model is divided in two parts. The first shows the actions performed by the driver and the mobile device she/he uses. In the second part, the remote assistance service, the user is a call center operator. *User actions* are carried out without interaction with the MCE system, while *interaction* indicates interaction between the user and the MCE system.

The monitoring software of the call center and the OBD in the car are not part of the system, but are important in the scenario. They are marked as *context collector* ((2) in Figure 2), their actions are integrated in the activity model and marked as being executed in the environment (actions with gray background). The notation allows specification of how context collectors are related to the actions in the activity model. In this diagram, they are used to carry out the action they are linked with.

The scenario starts when a problem with the car occurs (1). This can be either noticed by the driver or the OBD, which will show a warning then. The driver starts the ISHA, which gets the information of the OBD and analyzes it. If it does not have enough information, it checks whether it can contact the remote assistance over a (wireless) network (3). If there are connection possibilities, the gathered data is sent to the remote assistance system. The car manufacturer database is consulted resulting in the submission of instructions to the driver or the establishment of a call between the driver and an operator in the call center (4). Finally, the instructions are performed (5)³ by the driver resulting in a resolution of the problem or a failure which is logged to the incidents database.

Currently the notation shown in Figure 2 is only used in the design stage. A limited form of implementation generation, however, is planned for the near future. It is not our intention to generate program code, but to generate a set of declarative user interface descriptions that have links to the application logic and context sensing technologies. In order to generate such code, we will also

³The fork in this action indicates that another set of actions is triggered at this point.

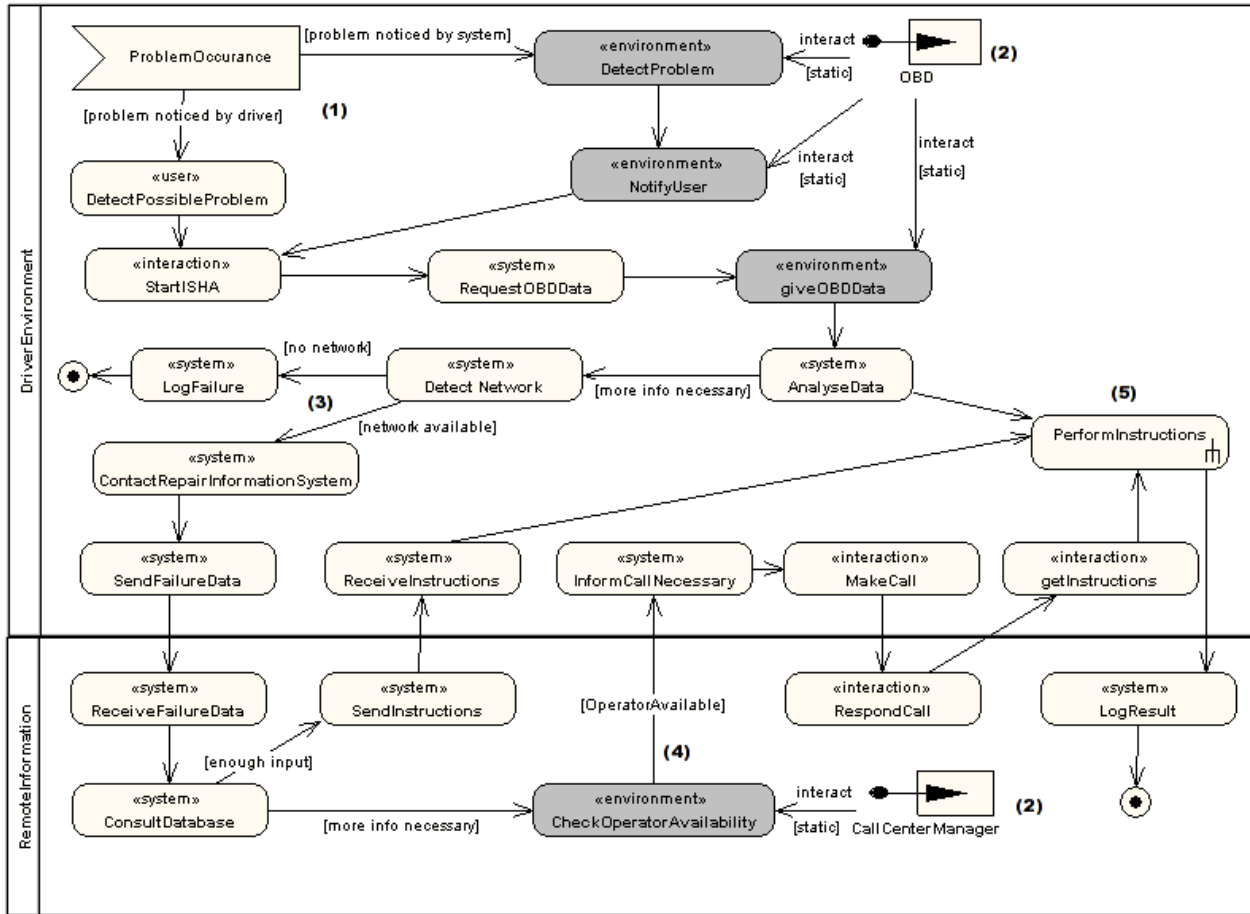


Figure 2: Activity diagram of the scenario

use diagrams that describe the application core, the context information and services, and the presentation of the user interface. Figure 3 shows this graphically; the models in the lower-half of the figure are used to generate information in the user interface specification in the upper half. The structure of the user interface specification in the figure corresponds to the structure of an XForms [6] document, which we also consider to be a possible target description format. The “context” and “application logic” sections of the document can be described in the XForms model sections, while the user interface structure can be described using XForms groups and controls.

6 Conclusion

We presented an overview of “in-car” interaction devices that can be used in the “driver self-help” scenario of the European Project MYCAREVENT. Each device has a set of properties that can be taken into account when designing an interactive system that uses these devices

for interaction. Properties such as support for communication protocols and graphical capabilities will influence the user interface that is presented to the user and the tasks that can be supported by these devices. The user interface engineer has to take into account the *context of use*, *specific requirements* of in-car interaction devices and the *role of the user* when designing an interface for the system required to support the scenario discussed here. Connectivity of the devices with other devices (connected with the car) or with an external network is part of the context of use. It is clear the user interface will differ depending on the context.

Starting from traditional modeling approaches we introduced a new notation for modeling context-sensitive interactive systems. This notation includes support for specifying contextual information and uses ideas from model-based user interface development to support multi-device user interface engineering. Because of its close integration with UML, different roles of different users

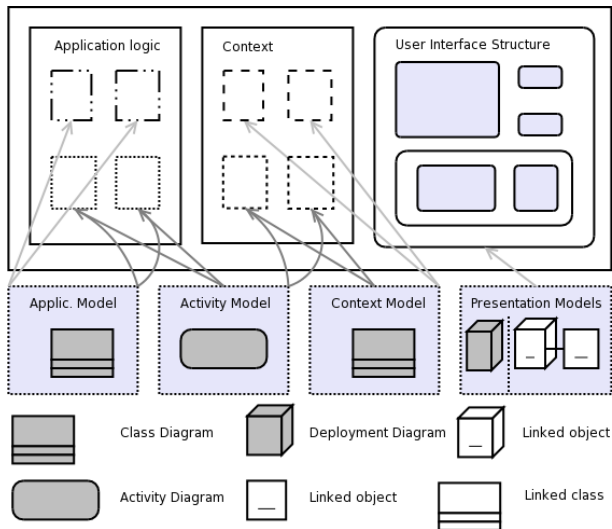


Figure 3: Generating user interface descriptions

could be expressed in the notation. Although the notation is based on earlier work in user interface design and built on UML, which has already proven its effectiveness, the effectiveness of the proposed notation still has to be validated using more rigorous testing by using it to model (a significant part of) a real-world application. The implementation of tool support that allows the generation of a partial user interface, based on models in the proposed notation, will also be very valuable in the evaluation of the notation and the approach that is taken.

Acknowledgements

We would like to thank the authors of the MY-CAREVENT use cases and Pilot scenario's, who provided us with valuable ideas. The research at the Expertise Centre for Digital Media (LUC) is partly funded by the Flemish government, EFRO (European Fund for Regional Development) and the Flemish Interdisciplinary Institute for Broadband Technology (IBBT). The MY-CAREVENT⁴ project FP6-IST No. 004402 is an Integrated Project sponsored by the European Commission in support of the Strategic Objective "Information Society Technologies (IST)" in the Sixth Framework Program.

References

- [1] Yamine Aït-ameur, Benoit Breholée, Patrick Girard, Laurent Guittet, and Francis Jambon. Formal Verification and Validation of Interactive Systems Specifications. In *IFIP 13.5 Working Conference on Human Error, Safety and Systems Development*, 2004.
- [2] Rémi Bastide, David Navarre, and Philippe Palanque. A tool-supported design framework for safety critical interactive systems. *Interacting with Computers*, 15:309–328, June 2003.
- [3] F. Bellotti, R. Berta, A. De Gloria, and M. Margarone. Developing in-car PDA-based tour guides. In Constantine Stephanidis and Julie Jacko, editors, *Human - Computer Interaction*, volume 2, pages 18 – 22, June 2003.
- [4] Jun-Kai Chiu and Sheue-Ling Hwang. Function Analysis and Control Panel Design of in-Car Computer Systems (E-Car). In Constantine Stephanidis and Julie Jacko, editors, *Human - Computer Interaction*, volume 2, pages 33 – 37, June 2003.
- [5] Tim Clerckx, Kris Luyten, and Karin Coninx. Generating Context-Sensitive Multiple Device Interfaces from Design. In Quentin Limbourg, Robert Jacob, and Jean Vanderdonckt, editors, *CADUI 2004*, volume 4. Kluwer Academic, 2004.
- [6] Micah Dubinko, Leigh L. Klotz, Roland Merrick, and T. V. Raman. Xforms 1.0. W3C, World Wide Web, <http://www.w3.org/TR/2003/REC-xforms-20031014/>, 2003.
- [7] Irene Mavrommati. Vehicle Navigation Systems: Case Studies from VDO Dayton. In Constantine Stephanidis and Julie Jacko, editors, *Human - Computer Interaction*, volume Volume II, pages 183 – 187, June 2003.
- [8] Fabio Paternò. *Model-Based Design and Evaluation of Interactive Applications*. Springer, 2000.
- [9] Fabio Paternò, Vincenzo Sabbatino, and Carmen Santoro. Using information in task models to support design of interactive safety-critical Applications. In *Proceedings of the working conference on Advanced visual interfaces*, pages 120–127. ACM Press, 2000.
- [10] Peter Rössger and Jörg Hofmeister. Cross Cultural Usability: An International Study on Driver Information Systems. In Constantine Stephanidis and Julie Jacko, editors, *Human - Computer Interaction*, volume 2, pages 253 – 257, June 2003.
- [11] Ian Sommerville. *Software Engineering*. Addison-Wesley, 7th edition, 2004.
- [12] Jan Van den Bergh and Karin Coninx. Towards modeling context-sensitive interactive applications. In *ACM Symposium on Software visualization*, Saint-Louis, Missouri, USA, May 14–15 2005. Accepted for publication.

⁴<http://www.mycarevent.com>